Lecture 5

OpenGL:

Control functions

OpenGL program structure

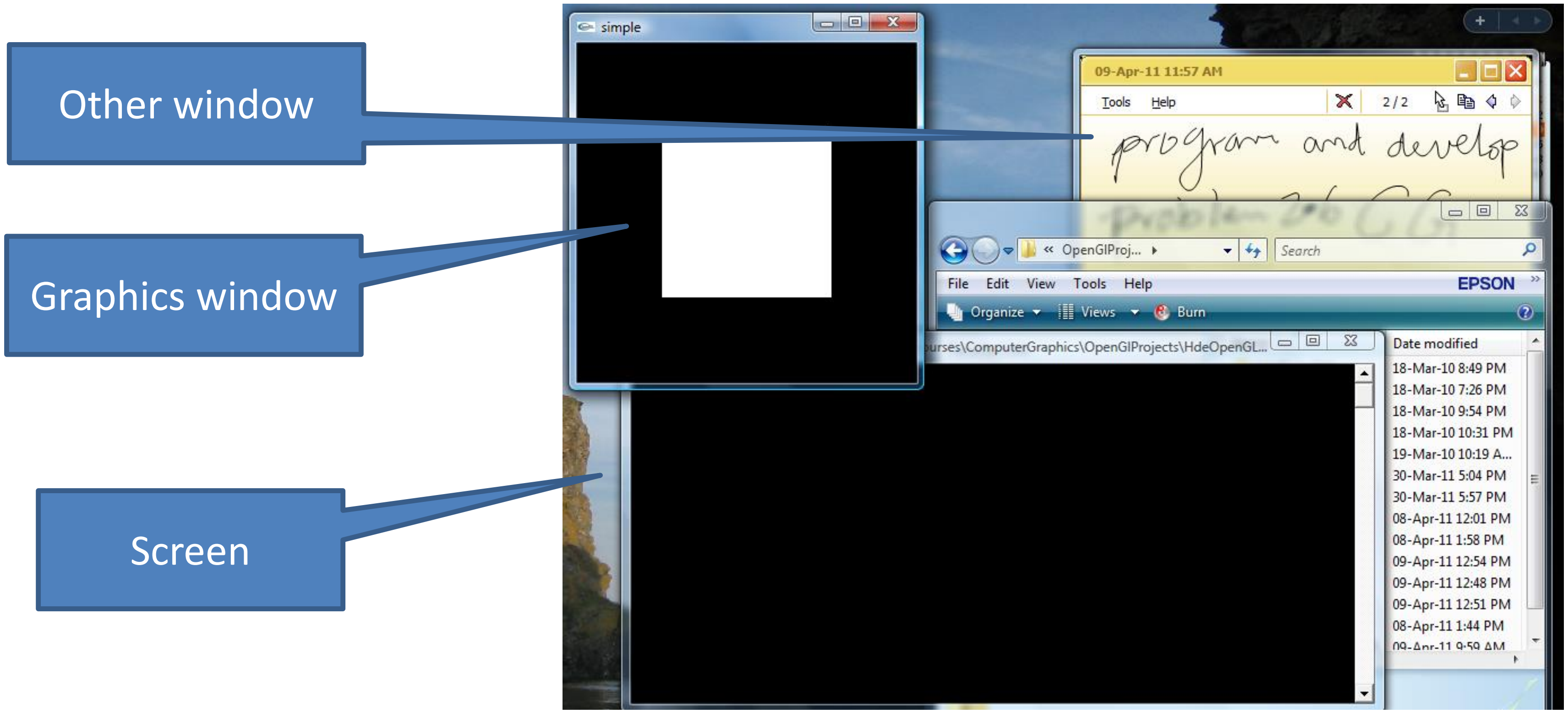
GLUT

- OpenGL Utility Toolkit (GLUT) is a library of functions that provides a simple interface between the OpenGL application and the underplaying systems (window system and operating system).
- It’s a cross-platform library (function naming and parameters are the same, but there are different implementations for different platforms)

Graphics window

- A graphics application displays it’s output on a rectangle area on the screen (window) and is controlled by the user using inputs from the keyboard, mouse, etc.
- Creating the window, manipulating it, getting inputs and related tasks are done through the GLUT
- A window is a rectangle area on the screen. It displays the contents of the Frame buffer. The position in the window is specified using window or screen coordinated (pixels)

Graphics window on a window system

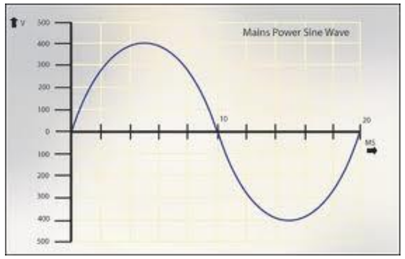


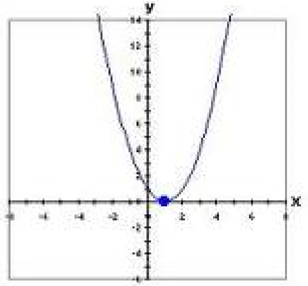
program and develop

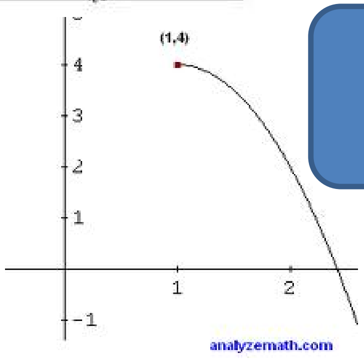
urses\ComputerGraphics\OpenGlProjects\HdeOpenGL...

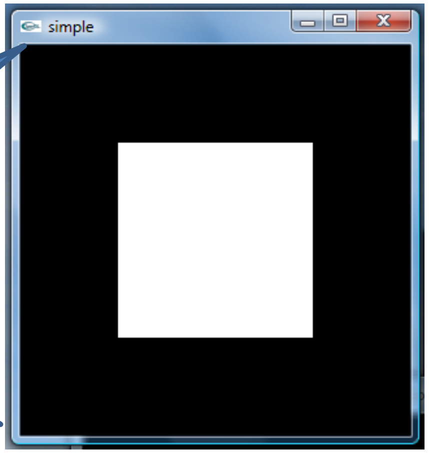
Date modified
18-Mar-10 8:49 PM
18-Mar-10 7:26 PM
18-Mar-10 9:54 PM
18-Mar-10 10:31 PM
19-Mar-10 10:19 A...
30-Mar-11 5:04 PM
30-Mar-11 5:57 PM
08-Apr-11 12:01 PM
08-Apr-11 1:58 PM
09-Apr-11 12:54 PM
09-Apr-11 12:48 PM
09-Apr-11 12:51 PM
08-Apr-11 1:44 PM
09-Apr-11 9:59 AM

Coordinates and positions









Window system origin

OpenGL origin

Engineers and scientists put the origin in a convenient location: Usually the center or the lower left corner
OpenGL consider the lower-left corner of the window as the origin (x increases right and y increases up)

Conversion is required

The origin of the window on a raster system is the upper-left corner (y increases down and x increases left.
Mouse input function gives the position in raster system coordinates

Interaction with the window system from a graphics application,

Initialize the window system, We can control the initialization using parameters

```
glutInit(int *argcv, char **argv)
```

Create window with a given title. The defaults for size, color model,.. Are used

```
glutCreateWindow(char *title)
```

But we can enforce our setting rather than using the defaults

```
glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
glutInitWindowSize(640, 480);
glutInitWindowPosition(0,0);
```

Although our screen may have a resolution of, say, 1280 x 1024 pixels, the window that we use can have any size. Thus, the frame buffer must have a resolution equal to the screen size. Conceptually, if we use a window of 300 x 400 pixels, we can think of it as corresponding to a 300 x 400 frame buffer, even though it uses only a part of the real frame buffer.

The defaults are GLUT_RGB rather than GLUT_INDEX, No hidden surface removal (not using GLUT_DEPTH), Single buffer (GLUT_SIGLE) rather that double buffering (GLUT_DOUBLE). Parameters are logically ORED Therefore, we did not use this before

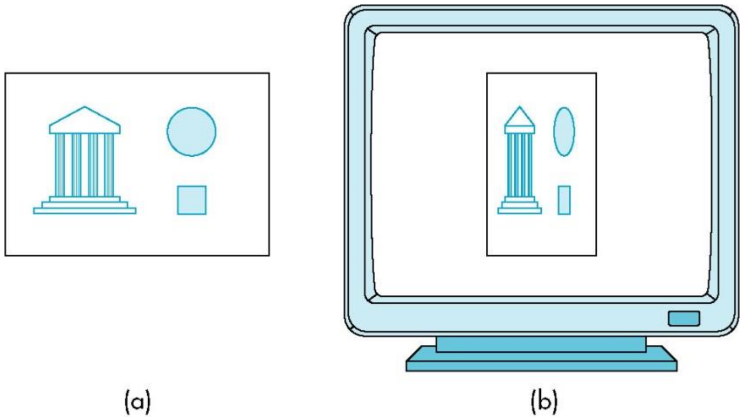
Aspect Ratio and Viewports

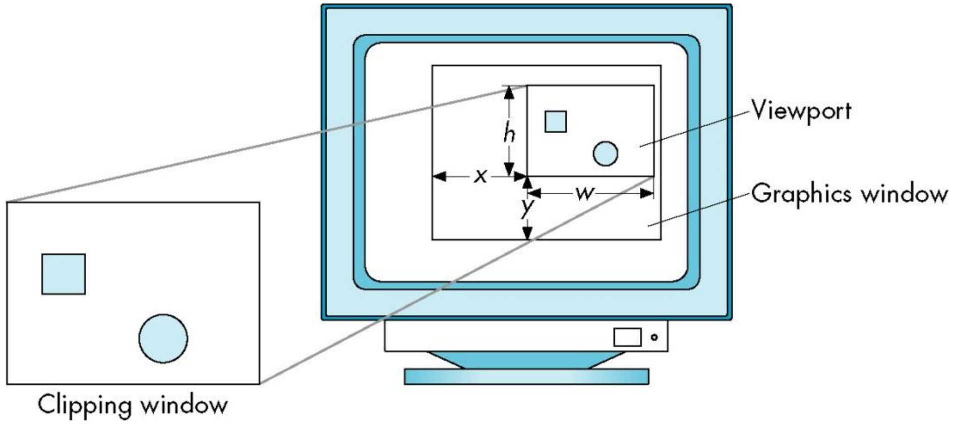
The aspect ratio of a rectangle is the ratio of the rectangle's width to its height.

The aspect ratio of the projection plane clipping window a (specified by glOrtho, for example) is independent of the aspect ration of the screen window (b)

Default: The entire contents of the projection plane clipping window is mapped to the entire screen window size (the view port is the entire window). Hence object deformation are possible.

We can avoid this deformation by making the two aspect ration the same either by controlling the clipping window size or by controlling the display size on the screen window (view port)



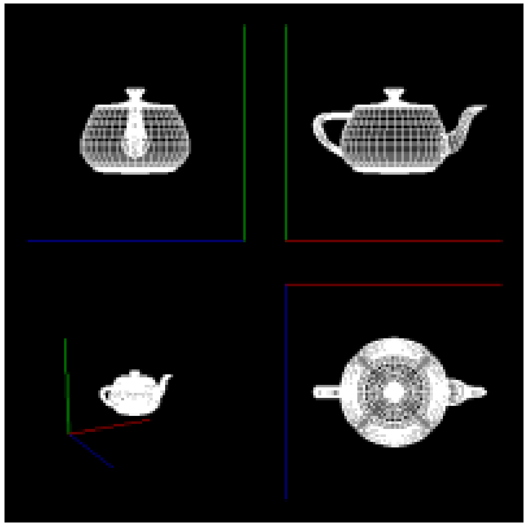


```
void glViewport(GLint x, GLint y, GLsizei w, GLsizei h)
```

x and y are the coordinate of the lower-left corner of the Viewport measured relative to the lower-left corner of the window. w and h are width and height. All is pixels

View port is a part of the GL state

The viewport is par t of the state. If we change the viewport between rendering objects or rendering the same objects with the viewport changed, we achieve the effect of multiple viewports with different images in different parts of the window.



There is a single viewport. Changing its location and size to different areas in the graphics screen window enables us to put different graphics on the same window

The main, Display, and Reshape Functions

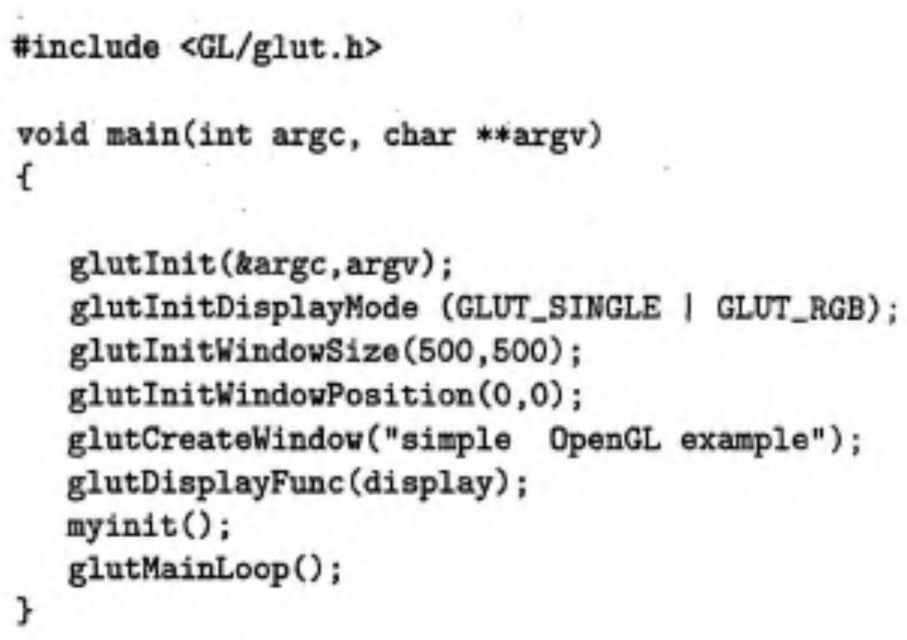
When the OpenGL programs executes. It renders primitives as it reads it.

- Why the programs waits after it finishes rendering and does not terminate immediately?
- What happens if the graphic widow is resized by the user, covered then uncovered by another window ?

When the OpenGL executes it starts by doing the required initialization, Creating the window, specifying some functions for specific purposes and finally enter a endless loop waiting for events to occur.
Functions and purposes

- Display function : responsible of drawing the graphics
- Reshape function: Responsible of taking the necessary actions when the screen window is resized

The main, Display, and Reshape Functions



```
#include <GL/glut.h>

void main(int argc, char **argv)
{

   glutInit(&argc,argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(0,0);
   glutCreateWindow("simple  OpenGL example");
   glutDisplayFunc(display);
   myinit();
   glutMainLoop();
}
```

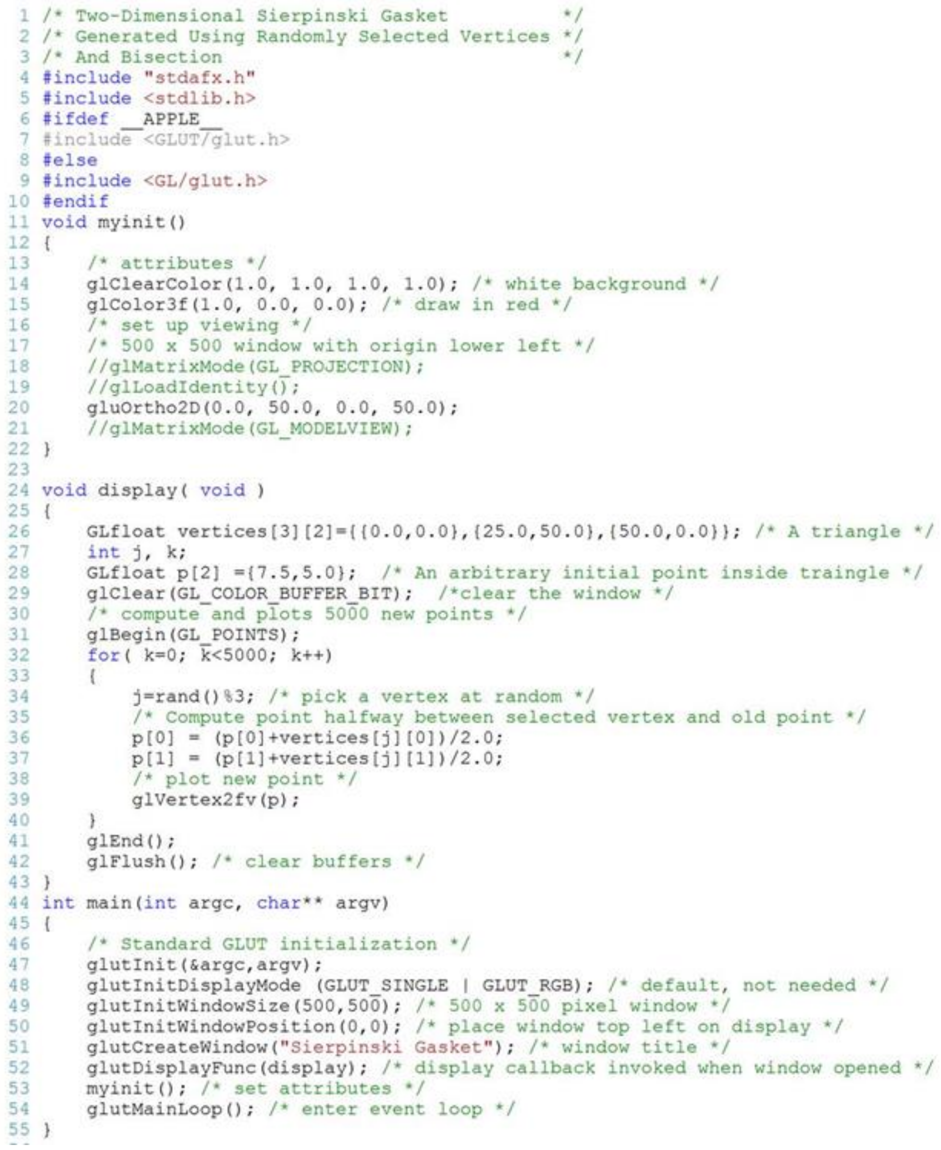
We need the two functions in the same source file or in an included file: One called display, the other myinit()

- The display function call back is required by OpenGL
- The myinit() function is just a preference to isolate the initialization of the state machine , viewing and other user options in a separate function

Program structure

This is the general structure that we will follow in almost all programs

- Display function has an arbitrary three vertices defined in it, and an arbitrary initial point is determined within the triangle that they define.
- Note that we picked the values in glOrtho so that the clipping window contains the whole triangle and there is no relationship between the size of the window on the display and the size of the clipping window; although by making both these windows square, there is no shape distortion of the gasket.



```c
/* Two-Dimensional Sierpinski Gasket          */
/* Generated Using Randomly Selected Vertices */
/* And Bisection                              */
#include "stdafx.h"
#include <stdlib.h>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
void myinit()
{
    /* attributes */
    glClearColor(1.0, 1.0, 1.0, 1.0); /* white background */
    glColor3f(1.0, 0.0, 0.0); /* draw in red */
    /* set up viewing */
    /* 500 x 500 window with origin lower left */
    //glMatrixMode(GL_PROJECTION);
    //glLoadIdentity();
    gluOrtho2D(0.0, 50.0, 0.0, 50.0);
    //glMatrixMode(GL_MODELVIEW);
}

void display( void )
{
    GLfloat vertices[3][2]={{0.0,0.0},{25.0,50.0},{50.0,0.0}}; /* A triangle */
    int j, k;
    GLfloat p[2] ={7.5,5.0};  /* An arbitrary initial point inside traingle */
    glClear(GL_COLOR_BUFFER_BIT);  /*clear the window */
    /* compute and plots 5000 new points */
    glBegin(GL_POINTS);
    for( k=0; k<5000; k++)
    {
        j=rand()%3; /* pick a vertex at random */
        /* Compute point halfway between selected vertex and old point */
        p[0] = (p[0]+vertices[j][0])/2.0;
        p[1] = (p[1]+vertices[j][1])/2.0;
        /* plot new point */
        glVertex2fv(p);
    }
    glEnd();
    glFlush(); /* clear buffers */
}
int main(int argc, char** argv)
{
    /* Standard GLUT initialization */
    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); /* default, not needed */
    glutInitWindowSize(500,500); /* 500 x 500 pixel window */
    glutInitWindowPosition(0,0); /* place window top left on display */
    glutCreateWindow("Sierpinski Gasket"); /* window title */
    glutDisplayFunc(display); /* display callback invoked when window opened */
    myinit(); /* set attributes */
    glutMainLoop(); /* enter event loop */
}
```

Report discussion

Previous lecture repot:
Problem 2.15

Next lecture repot:
Problem 2.9

In OpenGL, we can associate a color with each vertex. If the endpoints of a line segment have different colors assigned to them, OpenGL will interpolate between the colors as it renders the line segment. It will do the same for polygons. Use this property to display the Maxwell triangle: an equilateral triangle whose vertices are red, green, and blue.

We saw that a fundamental operation in graphics systems is to map a point (x, y) that lies within a clipping rectangle to a point (x_s, y_s) that lies in the viewport of a window on the screen. Assume that the two rectangles are defined by OpenGL function calls as follows:

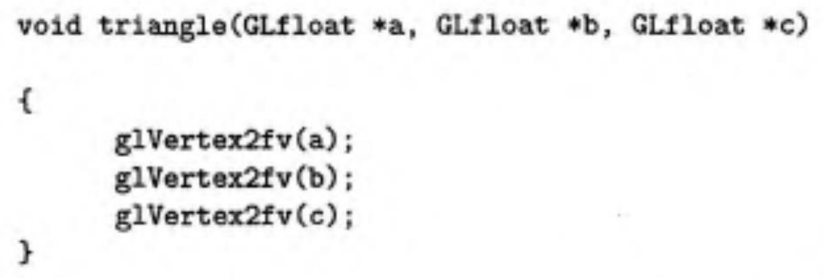
```
glViewport(u, v , w , h);
gluOrtho2D(x-min, x-max, y-min, y-max);
```

Find the mathematical equations that map (x, y) into (x_s, y_s).

Polygons and recursion: Gasket example

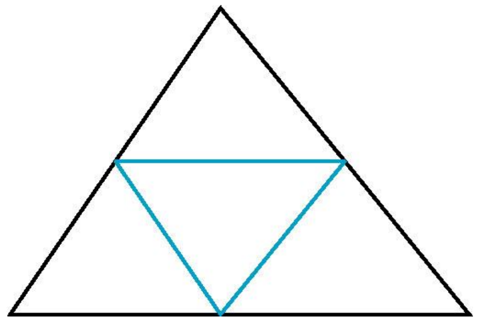
To structurally design an OpenGL program to produce a Gasket, we need to divide the task to simple tasks

First, we need a simple function that draw a triangle



```
void triangle(GLfloat *a, GLfloat *b, GLfloat *c)

{
        glVertex2fv(a);
        glVertex2fv(b);
        glVertex2fv(c);
}
```

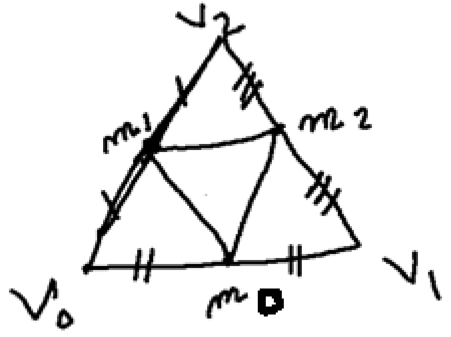


Gasket

- Connect the side mid-points in a triangle we sub divide it into 4 triangles
- Remove the middle triangle
- Repeat the process on each sub-triangle of the remaining three
- Continue until the triangle become very small

Polygons and recursion: Gasket example

Getting the sub-triangle vertices from the vertices of a triangle
Triangle vertices: Glfloat v[3][2];
The three mid-points are: Glfloat m[3][2];



```
for(j=0; j<2; j++) m[0][j]=(v[0][j]+v[1][j])/2.0;
for(j=0; j<2; j++) m[1][j]=(v[0][j]+v[2][j])/2.0;
for(j=0; j<2; j++) m[2][j]=(v[1][j]+v[2][j])/2.0;
```

Then we draw the triangles using the vertices or use them for further divisions

`(v[0], m[0], m[1]); (v[2], m[1], m[2]); and (v[1], m[2], m[0])`

Polygons and recursion: Gasket example

```
GLfloat v[3][2]={{-1.0, -0.58}, {1.0, -0.58}, {0.0, 1.15}};

int n;
```

```
void triangle( GLfloat *a, GLfloat *b, GLfloat *c)

/* specify one triangle */
{
        glVertex2fv(a);
        glVertex2fv(b);
        glVertex2fv(c);
}
```

```
void divide_triangle(GLfloat *a, GLfloat *b, GLfloat *c, int m)
{

/* triangle subdivision using vertex numbers */

    GLfloat v0[2], v1[2], v2[2];
    int j;
    if(m>0)
    {
        for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
        for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
        for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
        divide_triangle(a, v0, v1, m-1);
        divide_triangle(c, v1, v2, m-1);
        divide_triangle(b, v2, v0, m-1);
    }
    else triangle(a,b,c); /* draw triangle at end of recursion */
}
```

```
void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
    glMatrixMode(GL_MODELVIEW);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glColor3f(0.0,0.0,0.0);
}
```

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    divide_triangle(v[0], v[1], v[2], n);
    glEnd();
    glFlush();
}
```

```
int main(int argc, char **argv)
{
    n=5; /* or set number of subdivision steps here */
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Sierpinski Gasket");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```